Project 02: IMU Posture Sensor

CEN 598: Embedded Machine Learning

Erik Barraza Cordova

**System Design**

A screen shot of a computer program

Description automatically generatedThe motivation for my design stems from the provided “myreadimudemo”, which provided a monitoring method for the output of IMU sensors through the serial monitor. Due to the requirements of the project, I removed parts related to the magnetic field & instead focused on developing the outputs for IMU.accelerationAvailable() & IMU.gyroscopeAvailable(). Using the three axis output capabilities of the code provided, I decided to create a data collection system that would output into the Serial Monitor with modifications to this loop that allowed for the collection of data through a 4 second delay to allow me to move the sensor into the correct positions & to allow enough room between sensor readings so as to increase the likelihood of variability in the development of the sensor data. The sampling frequency was once every 4 seconds; therefore, the sampling frequency was 0.25 Hz. Separation between data point collection enables for there to be more variability in the data so as to make the rigid nature of thresholds for our defining of sensor measurements much more reflective of the variability and range of values that exist in real life.

A screenshot of a computer

Description automatically generatedA specific observation I had was that the iterative loop in this demo did not allow enough time for movement, so to adapt this code for my purposes of collecting data, I decided to increase the delay to 4000 ms (4 seconds) & implement millis() so as to monitor the time elapsed between data outputs. A challenge I had was keeping rhythm with the countdown as I gathered data for each posture, so to adjust for this, I adapted the RED builtin LED to blink for each second, allowing me to keep visual pace with the sensor between transitions. Prompting the data collection to csv was attempted. However, although more tedious for data collection/processing, the Serial Monitor Outputs in the demo above allowed for some manual work to be implemented in the collection of data. Alternatively, a secondary resource like PUTTY can be used to extract CSV data. I chose to maintain the serial.print() functions for both sensors, enabling me to collect the data from the serial monitor manually and saving into excel sheets for analysis.

In our system design for our real-time sensor, we built a heuristic algorithm based on the mean values of collected data in the accelerometer and gyroscope. This enables us to have defining thresholds for the data. However, in visualizing the data. We see that in those cases where there is a uniform distribution of values, taking the threshold value to be the mean will essentially cut out 50% of the values we have proven are outputted at the position. Mean values are a quick way to build a system that ignore the fact that there A green graph with numbers and a white background

Description automatically generatedare no precise value in the real world, and some variability is expected.

The distributions above are an example of the variability we see in the gyroscope settings for one of the postures. Although mostly experiencing fluctuations in close ranges on the Z distribution, the x and y distribution show large gaps and some outliers. This adds to the difficulties in using mean values for thresholds, as they can lead to cutting out large segments of the range that should be expected.

**Code Design – Pseudocode Framework**

# Pseudocode for Arduino Posture Detection using LSM9DS1 Sensor

*Define thresholds for each posture*

setup():

Initialize serial communication 9600 gaud

*Set LED pin as output*

Initialize the IMU sensor

If IMU initialization fails, enter an infinite loop

loop():

- If accelerometer and gyroscope data are available:

- Read data from the sensor

- Calculate the overall acceleration magnitude

- Check for different postures using sensor data

- If a posture condition is met:

- Display the detected posture

- Activate the LED with a specific blinking pattern

- If no posture condition is met:

- Indicate that the posture is "UNKNOWN"

- Keep the LED off

IF STATEMENTS:

isFaceDown

- Check if the acceleration is low

- Check if gyroscopic data indicates a low position

- If both conditions are met, it's considered "Face Down"

elseif

isFaceUp

- Check if the acceleration is high

- Check if gyroscopic data indicates a high position

- If both conditions are met, it's considered "Face Up"

elseif

isFaceSide

- Check if the acceleration is high

- Check if gyroscopic data indicates a sideways position

- If both conditions are met, it's considered "Face Side"

Else – UNKNOWN

**Checking for the name given in bools .>**

*If statements for blinkers*

*Executed and break*

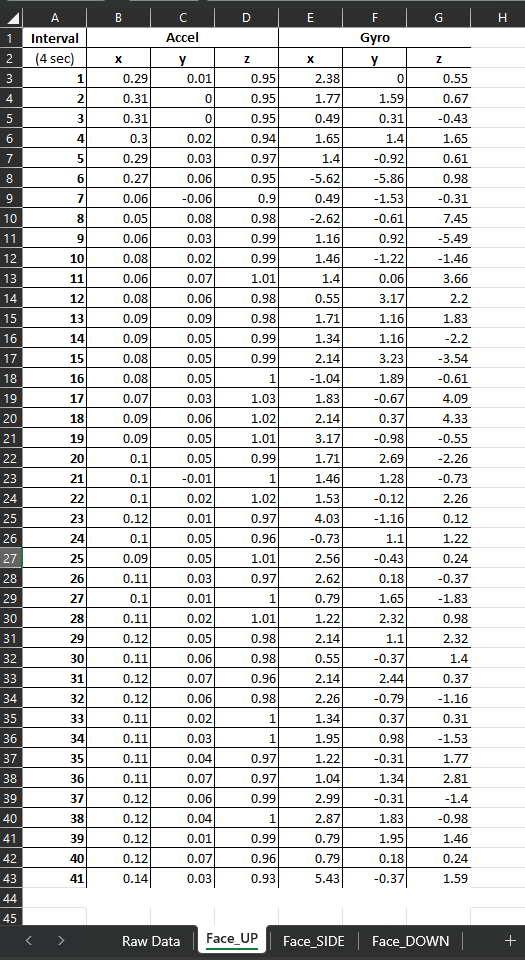
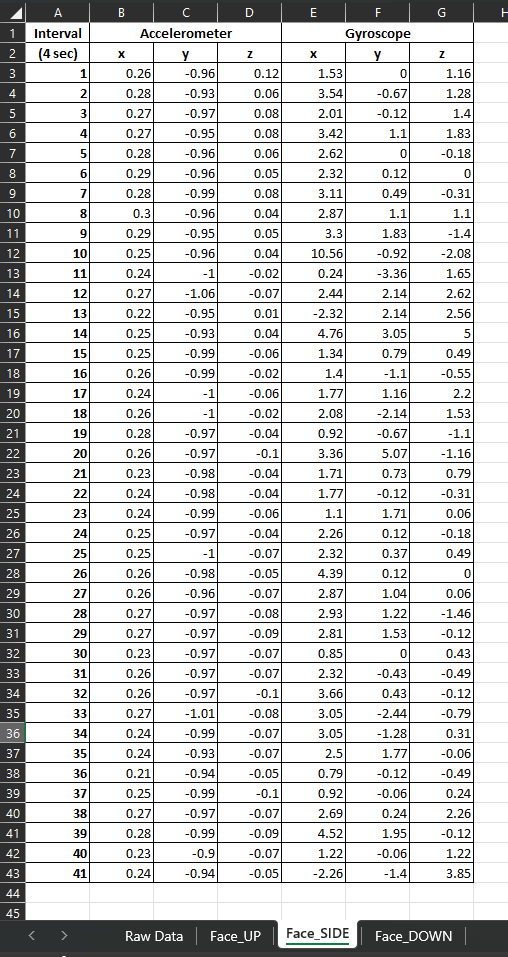
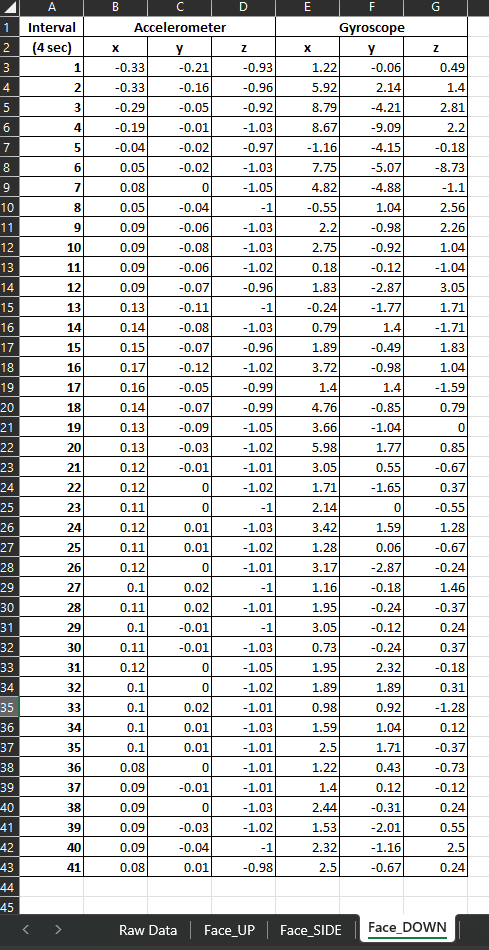
Difficulties I ran into with my Real Time Sensor Code (above) is the execution of conditions for the system. I initially began with defining cases for each of the postures, but was attempting to initialize & was unable to. Therefore, I attempted If statements that would run through the conditions quickly and allow for me to make comparisons and additional confirmations of both accelerometer and gyroscope data (with x,y,z for gyroscope), without difficulties. An assumption I made that lead to consolidating the values for x,y,z for acceleration was that it was non-direction, which required additional assistance from the gyroscope data to better perform an assessment. The same threshold finding steps were taken, taking the average of those distributions also are limiting, as there is also natural variability that is lost in defining parameters of threshold from the averages.

A computer screen shot of a program

Description automatically generated

**Algorithm (see pseudo above)**

An observation that lead to my design was the lack of overlap from Acceleration sensors data and Gyro data for the three postures. I thought if I can simply allow for the postures to be defined by these, we would have an easy time consolidating and enhancing the analysis of the three postures by using gyroscope data as well. Gyroscope data would provide orientation. Direction, which would enhance our analysis by providing a second feature to analyze on three separate planes (x,y,z). Operating criteria in parallel was my first assumption. My approach was not successful, I hyper parameterized using mean values instead of attempting to set ranges in a tight confidence interval around 60% around the mean. This would have been a more robust way of resolving my *rigidity of numbers* issues. My algorithm is more robus, but the amount of data & the lack of sensisibility to place value ranges for each sensor processing makes it less than useful for analyzing sensor data. The algorithm generally generates an output correctly every 3-4 seconds, with the rest indicating as “UNKNOWN”. The implementation of the gyroscope was supposed to help with analyzing orientation, but were rigid in the parameters that left no room for variability or outliers. Based in our data below, our algorithm did not yield successful tracking of the postures. Should we wish to build a better model, a lot more data would be useful as I only took to 41 data points for each posture after pruning data from the edges to account for transitions between postures.



**Results & Discussions**

**Acceleration Data**

A green graph with numbers and a white background

Description automatically generatedFace Up

A green graph with black text

Description automatically generatedFace – Side

A green graph with black text

Description automatically generatedFace Down

Unfortunately for this sensor case, we did not have a good accuracy, with less than 10% correct accuracy due to issues related to the parameters for the over defining of each posture. Gyroscopes pushed too many variables into each if statement & didn’t account for overlap or small variations that could completely tip the status. In addition, the if statement structure made is a rock slide if it didn’t fit into the strict parameters, making it fall into defaulting on UNKNOWN. Using Cases to enumerate my postures would’ve been a better way to define each. More data will also help resolve the problem, as I only took what was seen as statistically enough of a sample (41) so as to enable more time for development for each posture. With more hours I can build my thresholds on mountains more of data. The most difficult thing to accomplish was choosing how to best parameterize threshhhold, I do not believe my real time sensor worked in classifying the postures correctly.

References:

<https://www.researchgate.net/post/Minimum_sample_size_used_for_inferential_statistics>

1. <https://learn.adafruit.com/adafruit-arduino-lesson-5-the-serial-monitor/arduino-code>
2. <https://startingelectronics.org/software/arduino/learn-to-program-course/19-serial-input/>
3. <https://www.instructables.com/Advance-IO-With-Serial-Monitor/>
4. <https://docs.arduino.cc/built-in-examples/control-structures/SwitchCase2>
5. <https://www.programmingelectronics.com/tutorial-14-5-switch-case-statement-old-version/>
6. <https://www.deviceplus.com/arduino/the-basics-of-arduino-reading-switch-states/>
7. <https://www.youtube.com/watch?v=umZZjoyRbdw>
8. <https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-serial-monitor>
9. <https://forum.arduino.cc/t/hardware-switch-to-switch-the-software-serial-pins-being-transceived-on/525714>
10. <https://www.oreilly.com/library/view/arduino-cookbook/9781449399368/ch04.html>
11. <https://www.oreilly.com/library/view/arduino-cookbook/9781449399368/ch05.html>
12. <https://www.oreilly.com/library/view/getting-started-with/9781449379827/>
13. <https://wokwi.com/projects/322586637175358035>
14. <https://www.deviceplus.com/arduino/arduino-and-booleans-the-truth-is-greater-than-zero/>